Timsort

**Timsort** — гибридный алгоритм сортировки, сочетающий различные подходы.

Данный алгоритм является относительно новым и был придуман Тимом Петерсом. На массивах данных, которые содержат упорядоченные подмассивы, алгоритм Тима Петерса показывает себя намного лучше других сортировок. В настоящее время **Timsort** является стандартной сортировкой в **Python** и **GNU Octave**, реализован в **OpenJDK 7** и **Android JDK 1.5**.

**Содержание**

 [убрать]

* [1Основная идея алгоритма](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.9E.D1.81.D0.BD.D0.BE.D0.B2.D0.BD.D0.B0.D1.8F_.D0.B8.D0.B4.D0.B5.D1.8F_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0)
* [2Алгоритм](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.90.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC)
  + [2.1Обозначения](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.9E.D0.B1.D0.BE.D0.B7.D0.BD.D0.B0.D1.87.D0.B5.D0.BD.D0.B8.D1.8F)
  + [2.2Шаг 1. Вычисление minrun](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.A8.D0.B0.D0.B3_1._.D0.92.D1.8B.D1.87.D0.B8.D1.81.D0.BB.D0.B5.D0.BD.D0.B8.D0.B5_minrun)
  + [2.3Шаг 2. Алгоритм разбиения на подмассивы и их сортировка](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.A8.D0.B0.D0.B3_2._.D0.90.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC_.D1.80.D0.B0.D0.B7.D0.B1.D0.B8.D0.B5.D0.BD.D0.B8.D1.8F_.D0.BD.D0.B0_.D0.BF.D0.BE.D0.B4.D0.BC.D0.B0.D1.81.D1.81.D0.B8.D0.B2.D1.8B_.D0.B8_.D0.B8.D1.85_.D1.81.D0.BE.D1.80.D1.82.D0.B8.D1.80.D0.BE.D0.B2.D0.BA.D0.B0)
  + [2.4Шаг 3. Слияние](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.A8.D0.B0.D0.B3_3._.D0.A1.D0.BB.D0.B8.D1.8F.D0.BD.D0.B8.D0.B5)
  + [2.5Описание процедуры слияния](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.9E.D0.BF.D0.B8.D1.81.D0.B0.D0.BD.D0.B8.D0.B5_.D0.BF.D1.80.D0.BE.D1.86.D0.B5.D0.B4.D1.83.D1.80.D1.8B_.D1.81.D0.BB.D0.B8.D1.8F.D0.BD.D0.B8.D1.8F)
  + [2.6Пример](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.9F.D1.80.D0.B8.D0.BC.D0.B5.D1.80)
* [3Модификация процедуры слияния подмассивов (Galloping Mode)](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.9C.D0.BE.D0.B4.D0.B8.D1.84.D0.B8.D0.BA.D0.B0.D1.86.D0.B8.D1.8F_.D0.BF.D1.80.D0.BE.D1.86.D0.B5.D0.B4.D1.83.D1.80.D1.8B_.D1.81.D0.BB.D0.B8.D1.8F.D0.BD.D0.B8.D1.8F_.D0.BF.D0.BE.D0.B4.D0.BC.D0.B0.D1.81.D1.81.D0.B8.D0.B2.D0.BE.D0.B2_.28Galloping_Mode.29)
* [4Доказательство времени работы алгоритма](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.94.D0.BE.D0.BA.D0.B0.D0.B7.D0.B0.D1.82.D0.B5.D0.BB.D1.8C.D1.81.D1.82.D0.B2.D0.BE_.D0.B2.D1.80.D0.B5.D0.BC.D0.B5.D0.BD.D0.B8_.D1.80.D0.B0.D0.B1.D0.BE.D1.82.D1.8B_.D0.B0.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D0.B0)
* [5См. также](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.A1.D0.BC._.D1.82.D0.B0.D0.BA.D0.B6.D0.B5)
* [6Источники информации](https://neerc.ifmo.ru/wiki/index.php?title=Timsort#.D0.98.D1.81.D1.82.D0.BE.D1.87.D0.BD.D0.B8.D0.BA.D0.B8_.D0.B8.D0.BD.D1.84.D0.BE.D1.80.D0.BC.D0.B0.D1.86.D0.B8.D0.B8)

Основная идея алгоритма

Алгоритм **Timsort** состоит из нескольких частей:

* Начало.
* **Шаг 1**. Входной массив разделяется на подмассивы фиксированной длины, вычисляемой определённым образом.
* **Шаг 2**. Каждый подмассив сортируется [сортировкой вставками](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0%D0%BC%D0%B8), [сортировкой пузырьком](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC) или любой другой устойчивой сортировкой.
* **Шаг 3**. Отсортированные подмассивы объединяются в один массив с помощью модифицированной [сортировки слиянием](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D0%B5%D0%BC).
* Конец.

Рассмотрим теперь каждый шаг в отдельности.

Алгоритм

**Обозначения**

* nn — размер входного массива.
* runrun — подмассив во входном массиве, который обязан быть упорядоченным одним из двух способов:
  + строго по убыванию ai>ai+1>…ai>ai+1>….
  + нестрого по возрастанию ai⩽ai+1⩽…ai⩽ai+1⩽….
* minrunminrun — минимальный размер подмассива, описанного в предыдущем пункте.

**Шаг 1. Вычисление minrun**

* Начало.
* **Шаг 0**. Число minrunminrun определяется на основе nn, исходя из следующих принципов:
  + Не должно быть слишком большим, поскольку к подмассиву размера minrunminrun будет в дальнейшем применена сортировка вставками (эффективна только на небольших массивах).
  + Оно не должно быть слишком маленьким, так как чем меньше подмассив, тем больше итераций слияния подмассивов придётся выполнить на последнем шаге алгоритма. Оптимальная величина для nminrunnminrun — *степень двойки*. Это требование обусловлено тем, что алгоритм слияния подмассивов наиболее эффективно работает на подмассивах примерно равного размера.
  + Автором алгоритма было выбрано оптимальное значение, которое принадлежит диапазону [32;65)[32;65) (подробнее о том, почему так, будет сказано ниже).
  + Исключение: если n<64n<64, тогда n=minrunn=minrun и **Timsort** превращается в сортировку вставками.
* **Шаг 1**. Берем старшие 6 бит числа nn и добавляем единицу, если в оставшихся младших битах есть хотя бы один ненулевой.

Нетрудно понять, что после таких вычислений, nminrunnminrun будет равно степени двойки или немного меньше степени двойки.

* Конец.

**int** minRunLength(n):

flag = 0 // будет равно 1, если среди сдвинутых битов есть хотя бы один ненулевой

**while** (n ⩾⩾ 64)

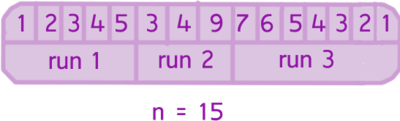
flag |= n & 1

n >>= 1

**return** n + flag

**Шаг 2. Алгоритм разбиения на подмассивы и их сортировка**

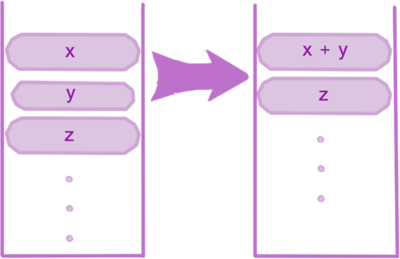
На данном этапе у нас есть входной массив, его размер nn и вычисленное число minrunminrun. Обратим внимание, что если данные изначального массива достаточно близки к случайным, то размер упорядоченных подмассивов близок к minrunminrun,. Но если в изначальных данных были упорядоченные диапазоны, то упорядоченные подмассивы могут иметь размер, превышающий minrunminrun.

[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:MinrunExample.png)

* Начало.
* **Шаг 0**. Указатель текущего элемента ставится в начало входного массива.
* **Шаг 1**. Начиная с текущего элемента, идет поиск во входном массиве упорядоченного подмассива runrun. По определению, в runrun однозначно войдет текущий элемент и следующий за ним. Если получившийся подмассив упорядочен по убыванию, то после вычисления runrun для текущего массива элементы переставляются так, чтобы они шли по возрастанию.
* **Шаг 2**. Если размер текущего runrun меньше minrunminrun, тогда выбираются следующие за найденным подмассивом runrun элементы в количестве minrun−size(run)minrun−size(run). Таким образом, на выходе будет получен подмассив размером большим или равным minrunminrun, часть которого (в лучшем случае — он весь) упорядочена.
* **Шаг 3**. К данному подмассиву применяем сортировку вставками. Так как размер подмассива невелик и часть его уже упорядочена — сортировка работает эффективно.
* **Шаг 4**. Указатель текущего элемента ставится на следующий за подмассивом элемент.
* **Шаг 5**. Если конец входного массива не достигнут — переход к шагу 1.
* Конец.

**Шаг 3. Слияние**

Нужно объединить полученные подмассивы для получения результирующего упорядоченного массива. Для достижения эффективности, нужно *объединять подмассивы примерно равного размера* и *cохранять стабильность алгоритма*.

[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Merge2mas.png)

* Начало.
* **Шаг 0**. Создается пустой стек пар << индекс начала подмассива, размер подмассива >>.
* **Шаг 1**. Берется первый упорядоченный подмассив.
* **Шаг 2**. Добавляется в стек пара данных << индекс начала текущего подмассива, его размер >>.
* **Шаг 3**. Пусть X,Y,ZX,Y,Z — длины верхних трех интервалов, которые лежат в стеке. Причем XX — это последний элемент стека (если интервалов меньше трёх, проверяем лишь условия с оставшимися интервалами).
* **Шаг 4**. Повторяем пока выражение (Z>X+Y∧Y>XZ>X+Y∧Y>X) не станет истинным
  + Если размер стека не меньше 22 и Y⩽XY⩽X — сливаем XX c YY.
  + Если размер стека не меньше 33 и Z⩽X+YZ⩽X+Y — сливаем YY c min(X,Z)min(X,Z).
* **Шаг 5**. Переходим к шагу 2.
* Конец

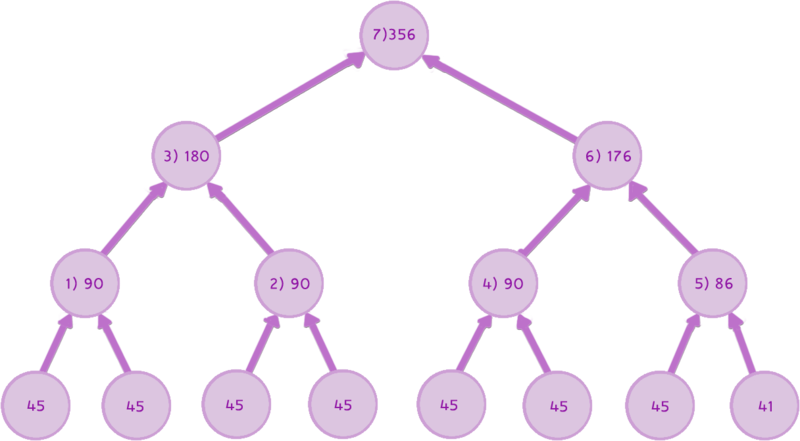
Основная цель этой процедуры — сохранение баланса. Изменения будут выглядеть как на картинке, а значит и размеры подмассивов в стеке эффективны для дальнейшей сортировки слиянием.

**Описание процедуры слияния**

* Начало.
* **Шаг 0**. Создается временный массив в размере меньшего из сливаемых подмассивов.
* **Шаг 1**. Меньший из подмассивов копируется во временный массив, но надо учесть, что если меньший подмассив — правый, то ответ (результат сливания) формируется справа налево. Дабы избежать данной путаницы, лучше копировать всегда левый подмассив во временный. На скорости это практически не отразится.
* **Шаг 2**. Ставятся указатели текущей позиции на первые элементы большего и временного массива.
* **Шаг 3**. На каждом шаге рассматривается значение текущих элементов в большем и временном массивах, берется меньший из них, копируется в новый отсортированный массив. Указатель текущего элемента перемещается в массиве, из которого был взят элемент.
* **Шаг 4**. Предыдущий пункт повторяется, пока один из массивов не закончится.
* **Шаг 5**.Все элементы оставшегося массива добавляются в конец нового массива.
* Конец.

**Пример**

Возьмем n=356n=356. При таком nn minrunminrun оказался равным 4545. Ниже представлена работа алгоритма. Числа с закрывающей скобкой показывают номера шагов, на которых произошло сливание нижестоящих подмассивов.

[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Example.png)

Модификация процедуры слияния подмассивов (Galloping Mode)

Рассмотрим процедуру слияния двух массивов:

A=1,2,3,…,9999,10000A=1,2,3,…,9999,10000

B=20000,20001,20002,…,29999,30000B=20000,20001,20002,…,29999,30000

Вышеуказанная процедура для них сработает, но каждый раз на её четвёртом пункте нужно будет выполнить одно сравнение и одно копирование. В итоге 1000010000 сравнений и 1000010000 копирований. **Timsort** предлагает в этом месте модификацию, которая получила называет **галоп**. Алгоритм следующий:

* Начало.
* **Шаг 0**. Начинается процедура слияния.
* **Шаг 1**. На каждой операции копирования элемента из временного или большего подмассива в результирующий запоминается, из какого именно подмассива был элемент.
* **Шаг 2**. Если уже некоторое количество элементов (например, в **JDK 7** это число равно 7) было взято из одного и того же массива — предполагается, что и дальше придётся брать данные из него. Чтобы подтвердить эту идею, алгоритм переходит в режим **галопа**, то есть перемещается по массиву-претенденту на поставку следующей большой порции данных бинарным поиском (массив упорядочен) текущего элемента из второго соединяемого массива.
* **Шаг 3**. В момент, когда данные из текущего массива-поставщика больше не подходят (или был достигнут конец массива), данные копируются целиком.
* Конец.

Для вышеописанных массивов A,BA,B алгоритм выглядит следующим образом: Первые 77 итераций сравниваются числа 1,2,3,4,5,6,71,2,3,4,5,6,7 из массива AA с числом 2000020000, так как 2000020000 больше, то элементы массива AA копируются в результирующий. Начиная со следующей итерации алгоритм переходит в режим **галопа**: сравнивает с числом 2000020000 последовательно элементы 8,10,14,22,38,7+2i−1,…,100008,10,14,22,38,7+2i−1,…,10000 массива AA (∼logn(∼log⁡n сравнений)). После того как конец массива AA достигнут и известно, что он весь меньше BB, нужные данные из массива AA копируются в результирующий.

Доказательство времени работы алгоритма

Не сложно заметить, что чем меньше массивов, тем меньше произойдёт операций слияния, но чем их длины больше, тем дольше эти слияния будут происходить. На малом количестве длинных массивов хорошо помогает вышеописанный метод **Galloping Mode**. Хоть он и не даёт асимптотического выигрыша, но уменьшает константу. Пусть kk — число кусков, на которые разбился наш исходный массив, очевидно kk = ⌈nminrun⌉⌈nminrun⌉.

Главный факт, который нам понадобится для доказательства нужной оценки времени работы в O(nlogn)O(nlog⁡n) — это то, что сливаемые массивы **всегда** имеют примерно одинаковую длину. Можно сказать больше: пока k>3k>3 сливаемые подмассивы будут именно одинаковой длины (данный факт хорошо виден на примере). Безусловно, после разбиения массива на блоки длиной minrunminrun последний блок может быть отличен от данного значения, но число элементов в нём не превосходит константы minrunminrun.

Мы выяснили, что при слиянии, длинна образовавшегося слитого массива увеличивается в ≈2≈2 раза. Таким образом получаем, что каждый подмассив runiruni может участвовать в не более O(logn)O(log⁡n) операций слияния, а значит и каждый элемент будет задействован в сравниниях не более O(logn)O(log⁡n) раз. Элементов nn, откуда получаем оценку в O(nlogn)O(nlog⁡n).

Также нужно сказать про [сортировку вставками](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0%D0%BC%D0%B8), которая используется для сортировки подмассивов runiruni: в нашем случае, алгоритм работает за O(minrun+inv)O(minrun+inv), где invinv — число обменов элементов входного массива, равное числу инверсий. C учетом значения kk получим, что сортировка всех блоков может занять O(minrun+inv)⋅k=O(minrun+inv)⋅O(minrun+inv)⋅k=O(minrun+inv)⋅⌈nminrun⌉⌈nminrun⌉. Что в худшем случае (inv=minrun(minrun−1)2)(inv=minrun(minrun−1)2) может занимать O(n⋅minrun)O(n⋅minrun) времени. Откуда видно, что константа minrunminrun играет немалое значение: при большом minrunminrun слияний будет меньше, а сортировки вставками будут выполняться долго. Причём эти функции растут с разной скоростью, поэтому и ещё после экспериментов на различных значениях и был выбран оптимальный диапазон — от 3232 до 6464.